LocAppl Flow Diagram

Commentary

Feb 17, 2000

LocAppl

This entry is called by Data Access Table processing when the 0x1D entry is encountered that means to call all enabled local applications. It does this job by calling LACall for each entry in the LATBL. It retains a copy of the enable bit status in the LATBL entry, so it can detect transitions that imply initialization or termination calls are required. After scanning through all LATBL entries, NewScan is called to free dynamic memory used by a currently-executing local application that has the new download flag set. On the next cycle, LACall will detect this situation and force a new copy to be allocated in dynamic memory to hold the new version.

LACall

After invoking ProgPtr, if a newly-downloaded program copy is detected, a termination call is forced in order to properly sequence to the new version. The enable Bit# is saved around the call to the local application, so that in event of a system abort occurring while the application is executing, the system abort routine can arrange to clear the enable bit. When the system reboots, the local application will not be reenabled to possible cause a repeated abort.

ProgPtr

This routine, given an 8-character program file name, returns a pointer to the executable code in dynamic memory. To do this, it searches the CODES table for a match (FindP); if necessary, it allocates a block of memory to hold the executable code, performing a checksum check in the process.

It has two variations on this theme. In the only variation that is used (by PgTitle), the CODES table is searched and the checksum is checked, but it does not create an executable copy of the code in dynamic memory. This variation is used to display the line on the index page that includes an indicator to show whether a valid program copy is available in the non-volatile memory. This check for validity includes examining the first word of the code. (We may not be able to do this check under VxWorks.)

As part of the examination of the CODES table entry, the size field is checked to be in the range of 32–32K words. This is done just before performing the checksum calculation, which is merely a longword sum of unsigned 16-bit words that comprise the program code. (The largest program size currently in use is about 10K words.) If the checksum check fails, the CODES table entry PtrD field is cleared, and the nonvolatile program file is freed. If the checksum check was successful, LoadP is called to bring the program into dynamic memory.

LoadP

Allocate dynamic memory sufficient to hold a copy of a program, copy into it from the nonvolatile memory, establish the executable pointer, and increment the diagnostic counter in the LATBL entry.

FindP

Look for a match on the 8-character file name and return a pointer to the matching CODES table entry. The three modules PReqDGen, ReqDGenP, and SetProg, make calls to FindP in the course of supporting access to non-volatile memory files.

ClrPtrE

Given a pointer to a CODES table entry, verify that the executable pointer field is the address of an allocated program block in dynamic memory; then free that block as well as the executable pointer field.

LAPtrX

Given a pointer to the parameter list in an LATBL entry, back up from there to find the name of the program, then search to find the corresponding CODES table entry, and finally return the executable pointer field. This is needed by ACReq, Serial, and SNAP, in order to call a local application. The linkage to the local application is made via an entry in the PROTO table that was established by a call to OpenPro. The function LAEntPro is used to obtain the pointer to an LATBL entry parameter list given a task name (or UDP port "task name"). ACReq uses LAPtrX for preparing to call an Acnet server LA, such as FTPM. Serial uses it to pass control to a serial port server LA, such as DNET that handles DeviceNET serial protocol access to PLC hardware. SNAP uses it to pass control to a UDP port server LA, such as ECHO.

InzLOOP

This routine is called at system initialization to coalesce all free blocks in non-volatile memory into one. This may not be possible under VxWorks. It calls the dreaded MSqueeze, which calls LAMove for each file that it needs to move.

NewScan

This routine is called once each cycle to check for newly-downloaded local application programs that are currently active. (See the description of LocAppl above.) For each that it finds, it frees the dynamically-allocated block that contains the executable code, thus forcing a reload on the following cycle of what may be a new and improved version of the LA. The effect is that the new version is immediately incorporated and brought into execution in place of the previous one.

ClrPage

For any page application that has been newly downloaded, free the executable area, if it exists. This routine is called by APTrig when any page application terminates. The logic does not support automatically switching to a new version of a page application after downloading; one must call up the page manually to use the new version.

VxWorks

Since we are using the real file system under VxWorks, we will not know where in non-volatile memory a given file resides. But since we won't be copying it into dynamic ram ourselves, it won't matter. Checksum checking will not apply. The pointer to downloadable memory should at least be nonzero and >= 2. We need to remove the more specific checks on the download pointer value that presently exist, or we can use a much larger constant value.

After loading a program into dynamic memory, analogous to the VxWorks "ld" command, we need to get the "pointer" to the entry point. This pointer is probably a pointer to a transition vector, but in any case, it should be the means of indirectly invoking the entry point in C.